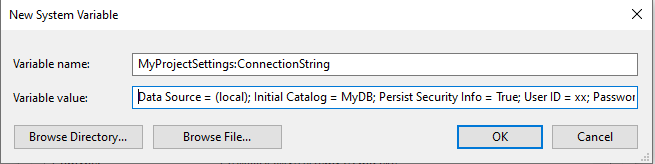
**Dynamic Application Configuration**

One of the problems that we face when moving code from one environment to another is configuration. On a student’s computer a project’s configuration settings are specific to their home environment. When this project is downloaded to a teacher’s computer, many of these configuration settings will have to change. A Database connection string is just one such example. To that end, I have an assembly that contains code to fix this problem.

The assembly you will need is called “ConfigurationAssistant”. It is designed to load configuration settings first from appsettings.json, and if configured to do so, override these settings with user secrets and/or environment settings. Which of these settings is used for your final configuration depends on your initial settings in appsettings.json. There are 3 entries that must appear in your appsettings.json file.

**UseUserSecrets**: If true, then it is expected that the project has created a user secret (using “Manage User Secrets”) project menu. The contents of the secrets.json file should be EXACTLY the same as the “MyProjectSettings” section of appsettings.json. The values in secrets.json will override the values found in appsettings.json.

**UseEnvironment**: If true, then any value from MyProjectSettings that you want to override should have an environment variable appropriately named. Naming for environment variables must include all the keys of the property that you are trying to override. For example, to override the “ConnectionString” value specified in MyProjectSettings, you would create the following SYSTEM environment variable:



**MyProjectSettings**:

The properties you place into the MyProjectSettings section of appsettings.json are strictly up to you. You can customize this any way you want. The code in ConfigurationAssistant will bind every property in the MyProjectSettings section into a property in an IUserConfiguration interface. Therefore, you must ensure that that Interface is structured EXACTLY like the MyProjectSettings section.

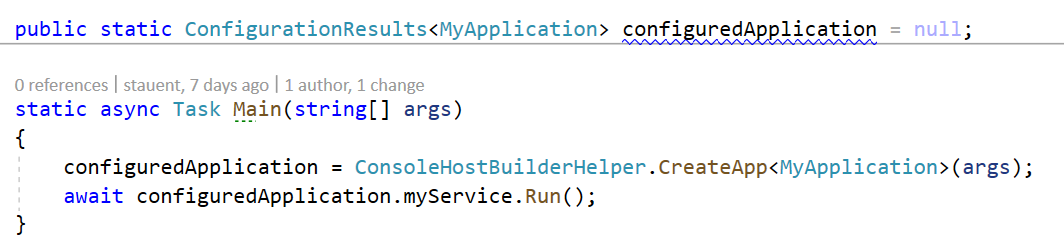


I have placed a few commonly needed property values in this section. If you don’t need them, you can leave then blank. I’ve found that almost every application needs a database connection string, and any application connecting to a web service needs some kind of login information from the client. With this in mind, the initial interface provided for you to access your configuration looks like this:

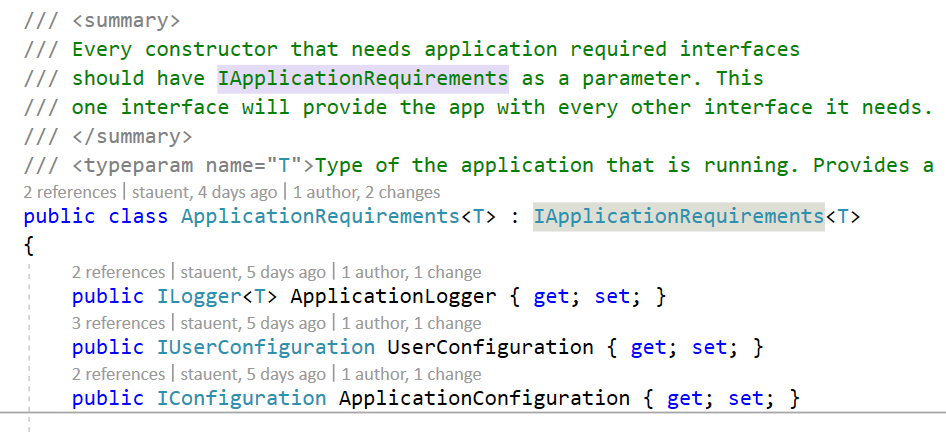


Remember that you can modify this interface to include ANYTHING you want. This is just a starting point.

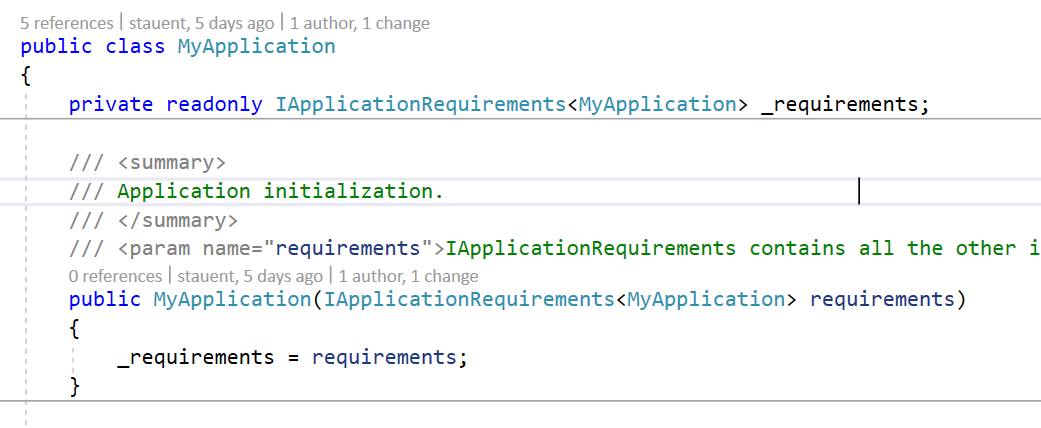
The sample application “CustomConfiguration” allows you to play around with the settings to see how it works. Here’s the code to set up your console application to use dependency injection. It will automatically load your configuration from appsettings.json/secrets.json correctly and produce an IUserConfiguration interface that you can inject in any constructor.



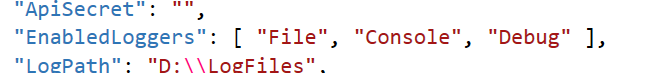
Because there are a few interfaces that are commonly used, I created the IApplicationRequirements<T> interface that is used to contain several other interfaces. This way you only have to inject one interface into your application and just use the contained interfaces where you need them.



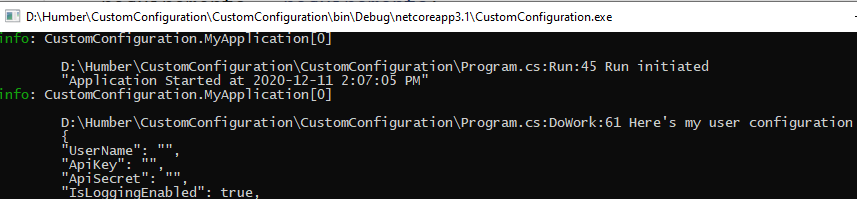
Here’s an example of how to inject this interface for use in your application:



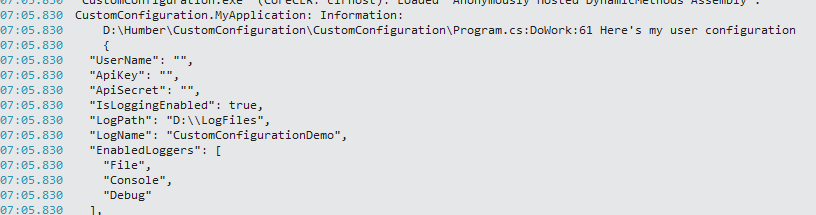
One additional feature that is configured for you in the IOC (Inversion of control container) is logging. It is very common to log your activity as your code runs to help you debug your application. There are 3 loggers available. The first is a console logger, then a debug logger and finally a file logger. Which loggers you have enabled for your application is controlled by the “EnabledLoggers” setting in your appsettings.json/secrets.json file:



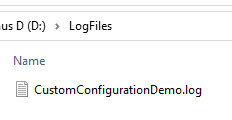
As you can see, I have enabled all 3 loggers by default. The only other option is “None”. If you specify “None” as the only option, or include it with the others, then NO logging will be performed. Here’s the output in the console window:

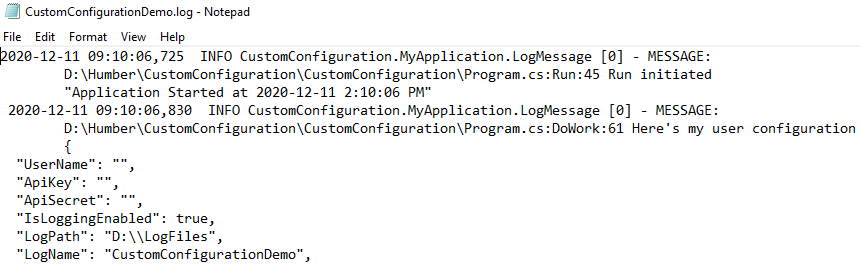


Here’s the debug (output) window in Visual Studio:

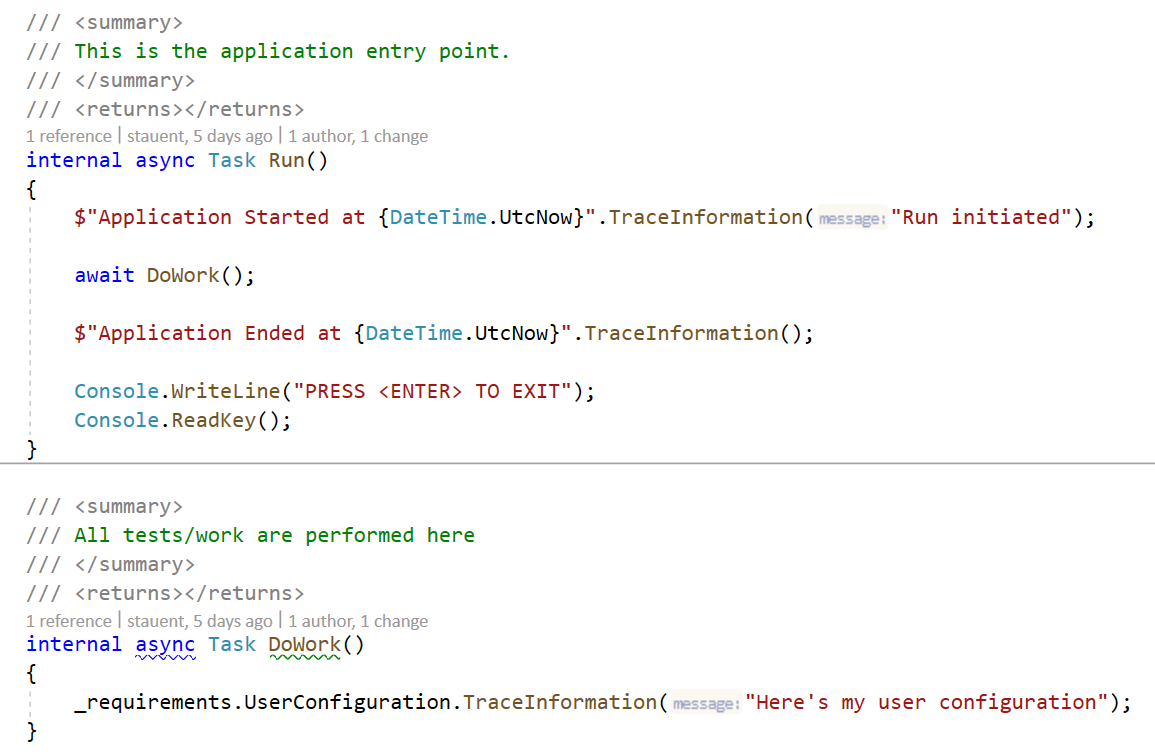


And here’s the output produced in the log file. Your appsettings has 2 entries that control where the log file is created. The “LogPath” and “LogName” control the location.





A number of extension methods have been created to make it very simple for you to log ANYTHING in your application. If you start writing “. Trace” after ANY variable, you’ll be presented with one of the extension methods that allow you to log your information. In the CustomConfiguration demo application, here’s the code that produces the output:



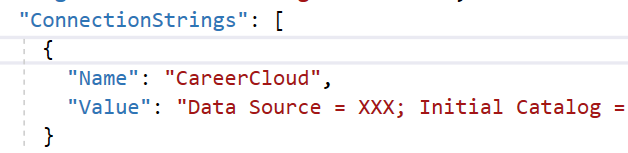
You’ll see that I’ve used the .TraceInformation extension method to display messages about starting and stopping the application, and also in DoWork() I log the entire contents of the user configuration interface. The format of the logged information is dictated by the setting:



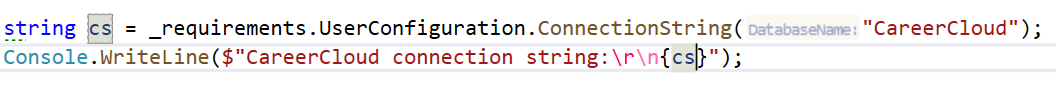
The default format is “Json”. In this case any object that is logged is serialized into a formatted Json string and then displayed. The only other option is “String”. In this case “ToString” is called on the object being logged. If you have the ToString() method overloaded, then this is the format that will be used. If you don’t have it overloaded, then whatever the default ToString() method does for that object is what will be used.

**ConnectionStings:**

To get any connection string that is part of your configuration, simply ask for it by the “Name” property you specified in your configuration. For example, if you had the following configuration:



...you would retrieve the connection string for “CareerCloud” using the following code:



If you DON”T have user secrets enabled, then you will retrieve the connection string from appsettings.json that looks like this:

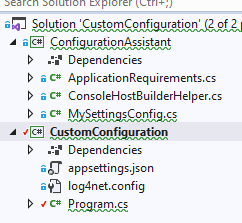
"Data Source = XXX; Initial Catalog =

If you DO have use secrets enabled, then you will get the valid connection string that works on your local machine. On my machine I get back a string that looks like this:

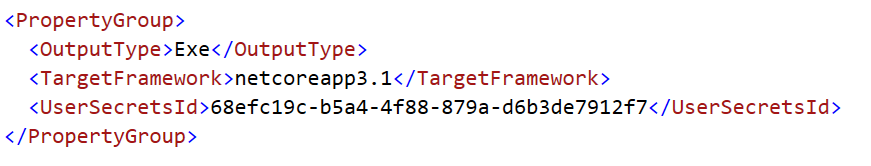
"Data Source = OPTIMUS; Initial Catalog =

It is important that you enable user secrets for all your applications that require database access. This way your code can be shared with anyone and your local connection string settings will not impact anyone.

To enable user secrets, double click your main project in the solution explorer. This will open the .csproj file:

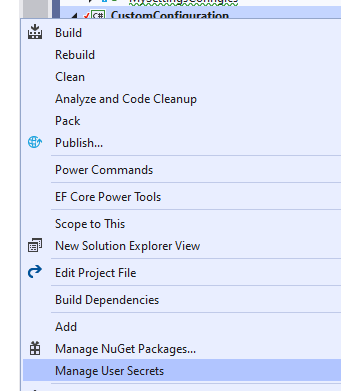


Any application that needs to use the same user secrets must have the identical <UserSecretsId> section installed in their .csproj file.

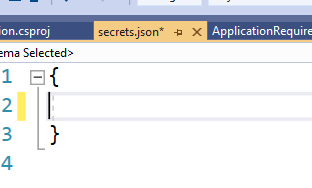


Any application that has the SAME <UserSecretsId> section will access the SAME secrets.json file. In this way you can set up your connection string information once, and use it across multiple applications.

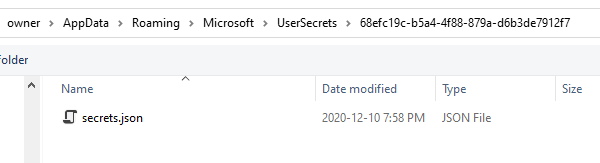
To create/edit your secrets.json file, simply right click on the project heading in solution explorer and select “Manage User Secrets”



You will immediately be presented with an empty secrets.json file



This file will physically exist at the following location:



The folder name is the same value that you had in your <UserSecretsId> section in the .csproj file:

68efc19c-b5a4-4f88-879a-d6b3de7912f7

Now copy the “MyProjectSettings” section from your appsettings.json file and simply change the connection string value to be something that is valid for your machine:



Save the secrets.json file and run the application. You SHOULD see the values from secrets.json appear instead of what is in your appsettings.json.